METHOD FOR ILLUSTRATING ARROW LOGIC RELATIONSHIPS BETWEEN GRAPHIC OBJECTS USING GRAPHIC DIRECTIONAL INDICATORS

REFERENCE TO RELATED APPLICATIONS

[0001] This application is a continuation-in-part of application serial no. 09/880,397, filed June 12, 2001, which is a continuation-in-part of application serial no. 09/785,049, filed on February 15, 2001, for which priority is claimed. The entireties of the prior applications are incorporated herein by reference.

FIELD OF THE INVENTION

[0002] The invention relates generally to computer operating environments, and more particularly to a method for performing operations in a computer operating environment.

BACKGROUND OF THE INVENTION

[0003] Graphic control devices, such as faders, buttons and dials, used in various computer applications are usually preprogrammed to perform designated functions. In some computer applications, a graphic control device may be programmed to control a graphic item, such as a graphic tool. Thus, there is an assigned relationship between the graphic control device and the graphic item. Usually, the graphic control device is labeled with some text that describes the function of that device, and the graphic item being controlled by that graphic control device may also be labeled with some text that identifies the graphic item. In these computer applications, assuming that multiple graphic control devices and items are on the screen, there is no means to determine which graphic item is being controlled by which graphic control device, unless each graphic control device is manipulated to see the effect on one of the graphic items.

[0004] Therefore, what is needed is a method for illustrating assigned relationships between graphic objects, e.g., between a graphic control device and a graphic item, in a computer environment.

SUMMARY OF THE INVENTION

[0005] A method for illustrating assigned relationships between graphic objects utilizes graphic directional indicators to show the assigned relationships between the objects. The assigned relationships may be arrow logic relationships, such as control arrow logic relationships. The graphic objects may be any type of graphic objects including graphic control devices. The graphic directional indicators may be arrows that are drawn between the graphic objects. The direction of the arrows can be used to indicate the sources and targets of the assigned relationships. The color of the arrows can also be used to indicate the type of assigned relationships.

[0006] A method in accordance with an embodiment of the invention includes displaying first and second graphic objects having an assigned relationship between them, and displaying a graphic directional indicator between the first and second graphic objects in response to a user input requesting the assigned relationship between the objects be shown.

[0007] An embodiment of the invention includes a storage medium, readable by a computer, tangibly embodying a program of instructions executable by the computer to perform the method steps for illustrating assigned relationships between graphic objects.

[0008] Other aspects and advantages of the present invention will become apparent from the following detailed description, taken in conjunction with the accompanying drawings, illustrated by way of example of the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] Figure 1 is a depiction of various color values of arrow components of the arrow logic system of the present invention.

[0010] Figure 2 is a depiction of various styles of arrow components of the arrow logic system.

[0011] Figure 3 is a screen shot of a portion of a possible Info Window for the arrow logic system.

[0012] Figure 4 is a depiction of a gradient fill arrow component of the arrow logic system.

[0013] Figure 5 is a depiction of an arrow color menu bar, hatched to indicate various colors and associated functions that may be selected.

[0014] Figure 6 is a depiction of an arrow menu bar, showing various colors and arrow styles that may be selected.

[0015] Figure 7 is a depiction of a copy arrow and the placement of the new copy of the existing object relative to the head of the copy arrow.

[0016] Figure 8 is a depiction of another copy arrow and the placement of the new copy of the existing star object relative to the head of the copy arrow.

[0017] Figure 9 is a depiction of a copy arrow and the placement of the new copy of the existing triangle object relative to the head of the copy arrow.

[0018] Figure 10 is a chart of arrow styles indicating the association of various copy transactions with their respective arrow styles. Most importantly, this indicates a user's ability to type, print, write or use a vocal command to reassign an arrow logic to a hand drawn arrow, by using arrow logic abbreviations.

[0019] Figure 11 is a depiction of a hand drawn arrow conveying the transaction of placing a list of music files inside a folder.

[0020] Figure 12 is a depiction of a hand drawn arrow conveying the transaction of selecting and placing a group of on-screen objects inside an on-screen object.

[0021] Figure 13 is a depiction of a hand drawn arrow conveying the transaction of selecting and placing a group of on-screen devices and/or objects inside an on-screen object.

[0022] Figure 14 is a depiction of another graphical method of using a hand drawn arrow to convey the transaction of selecting and placing a group of on-screen devices and/or objects inside an on-screen object.

[0023] Figure 15 is a depiction of a hand drawn arrow conveying the transaction of selecting and placing a list of music files inside a folder, where the selected and placed file names become grayed out.

[0024] Figure 16 is a depiction of an arrow directing a signal path from a sound file to an on-screen object which may represent some type of sound process.

[0025] Figure 17 is a depiction of the use of multiple arrows to direct a signal path among multiple on-screen devices and/or objects.

[0026] Figure 18 is a depiction of two arrows used to direct a send/sum transaction from two on-screen controllers to a third on-screen controller.

[0027] Figure 19 is a depiction of another example of two arrows used to direct a send/sum transaction from two on-screen controllers to a third on-screen controller.

[0028] Figure 20 is a further depiction of two arrows used to direct a send/sum transaction from two on-screen controllers to a third on-screen controller.

[0029] Figure 21 is another depiction of two arrows used to direct a send/sum transaction from two on-screen controllers to a third on-screen controller.

[0030] Figure 22 is a depiction of an arrow used to select and change a group of on-screen objects to another object.

[0031] Figure 23 is a depiction of an arrow used to select and change a property of multiple on-screen objects.

[0032] Figure 24 is a depiction of an arrow used to modify a transaction property of a previously drawn arrow.

[0033] Figure 25 is a depiction of an arrow used to apply the function of an on-screen controller to a signal being conveyed by another arrow to another on-screen object.

[0034] Figure 26 is a depiction of one technique for labeling an on-screen object with a word or phrase that imparts recognized functional meaning to the object.

[0035] Figure 27 is a depiction of another technique for labeling an on-screen object with a word or phrase that imparts recognized functional meaning to the object.

[0036] Figure 28 is a depiction of a further technique for labeling an on-screen object with a word or phrase that imparts recognized functional meaning to the object.

[0037] Figure 29 is another depiction of a technique for labeling an on-screen object with a word or phrase that imparts recognized functional meaning to the object.

[0038] Figures 30 and 31A are depictions of arrows used to define the rotational direction of increasing value for an on-screen knob controller.

[0039] Figure 31B is a depiction of the context of arrow curvature concentric to a knob, the context being used to determine which knob is associated with the arrow.

[0040] Figures 32 and 33 are depictions of arrows used to define the counter-default direction for an on-screen knob controller.

[0041] Figure 34A is a depiction of an arrow used to apply a control function of a device to one or more on-screen objects.

[0042] Figure 34B is a depiction of arrows used to apply control functions of two devices to the left and right tracks of an on-screen stereo audio signal object.

[0043] Figure 35 is a depiction of an arrow used to reorder the path of a signal through an exemplary on-screen signal processing setup.

[0044] Figure 36 is another depiction of an arrow used to reorder the path of a signal through an exemplary on-screen signal processing setup.

[0045] Figure 37 is a further depiction of an arrow used to reorder the path of a signal through an exemplary on-screen signal processing setup.

[0046] Figure 38 is a depiction of an arrow used to reorder the path of a signal through multiple elements of an exemplary on-screen signal processing setup.

[0047] Figure 39 is a depiction of an arrow used to generate one or more copies of one or more on-screen objects.

[0048] Figure 40A is a depiction of a typical double-ended arrow hand-drawn on-screen to evoke a swap transaction between two on-screen objects.

[0049] Figure 40B is a depiction of a double ended arrow created on the on-screen display to replace the hand drawn entry of Figure 40A.

[0050] Figure 41 is a depiction of an arrow hand-drawn on-screen.

[0051] Figure 42 is a depiction of a single-ended arrow created on-screen to replace the hand drawn entry of Figure 41.

[0052] Figure 43 is a depiction of a text entry cursor placed proximate to the arrow of Figure 42 to enable placement of command text to be associated with the adjacent arrow.

[0053] Figure 44 is a depiction of an arrow drawn through a plurality of objects to select these objects.

[0054] Figure 45 is a depiction of a line without an arrowhead used and recognized as an arrow to convey a transaction from the leftmost knob controller to the triangle object.

[0055] Figure 46 is a depiction of non-line object recognized and used as an arrow to convey a transaction between screen objects.

[0056] Figure 47A is a depiction of three graphic faders.

[0057] Figure 47B is a depiction of "control" arrows drawn between the faders of Figure 47A to assign "control" arrow logics to the faders.

[0058] Figure 47C is a depiction of the faders of Figure 47A after the "control" arrow logics have been assigned to the faders.

[0059] Figure 48 is a depiction of an Info Canvas object for one of the faders of Figure 47A.

[0060] Figure 49A illustrates the appearance of a display arrow when the "show arrow" command is entered for the first fader of three faders of Figure 47C.

[0061] Figure 49B illustrates the appearance of two display arrows when the "show arrow" command is entered for the second fader of three faders of Figure 47C.

[0062] Figure 49C illustrates the appearance of a display arrow when the "show arrow" command is entered for the third fader of three faders of Figure 47C.

[0063] Figure 50 is a flowchart of a process for showing one or more display arrows to illustrate arrow logics for a given graphic object.

[0064] Figure 51A is a flowchart of a routine to determine whether the object has displayable links in the process for showing one or more display arrows.

[0065] Figure 51B is a flowchart of a routine to show a display arrow in the process for showing one or more display arrows.

[0066] Figure 52 is a flowchart of a process called in the arrow logic display object when the delete command is activated for the display object.

[0067] Figure 53 is a flow diagram of a method for illustrating assigned relationships between graphic objects in accordance with an embodiment of the invention.

DETAILED DESCRIPTION

[0068] The fundamental concept of arrow logic operations is to draw arrows, or their equivalents, that command transactions to take place between two or more objects presented in or on a graphic display. Moreover, arrow logics provide visually distinct arrows that designate a particular transaction to correspond to each type of arrow. The visible trait of an arrow that is most easily distinguished in a graphic display is color. The assignment of a transaction to a specific color arrow can be carried out in the Global Info Window for arrows. It is also possible to make such an assignment directly in the Arrow Logic Info Window (or its equivalent) or directly on the arrow itself, by entering an arrow logic abbreviation command or its equivalent. (Note that the system provides info windows for objects <u>and</u> for actions.) Once these arrow logic categories are understood, there are a variety of ways of assigning specific category functions to arrows with specific colors.

[0069] In the arrow logic system, an arrow drawn on the screen conveys a transaction from the tail of the arrow to the head of the arrow. There is a wide range of arrow transactions, including transactions that extend between screen objects, as well as transactions that apply to a single screen object. When the arrow is drawn on the screen, the operation does not happen immediately. Instead, the arrow and/or object(s) that the arrow pertains to may flicker or change color or anything else that provides an easy to recognize visual change, and will continue to flicker until it is touched (by cursor contact, direct touch on the screen, stylus tap, or the like). Then the flickering stops, the arrows disappear (or return to their original graphical properties, i.e., color, shape, status, location, etc. and remain visible on the screen) and the transaction is completed.

[0070] The following is a partial list of transactions that may be carried out using arrow logics.

(a) Copy/Replace or Copy/Replace/Delete from screen

(b) Place Inside

(c) Send the signal or contents to

(d) Change to

(e) Specialty Arrows, which are determined by context, not necessarily by color.

(f) Insert an action or function in the stem of an arrow.

(g) Rotational direction for a knob

(h) Apply the control of a device to one or more objects, devices or text.

(i) Reorder or redirect a signal path among screen objects.

(j) Create multiple copies of, and place these copies on the screen display.

k) Swap

[0071] **<u>Utilizing different colors for different arrow transactions.</u>**

[0072] The arrow logics system provides different techniques for assigning arrow colors to particular transactions, in order to accommodate different amounts of flexibility and complexity according to how much each individual user can manage, according to his or her level of experience. The following ways of assigning colors start with the simplest way to utilize arrow Exchange logics and become increasingly more flexible and complicated.

a. Lower level user: Assign one arrow color per arrow logic category. With this approach, for each of the above six arrow logic categories, only one color would be used. This requires that only one type of arrow color per category can be used. For instance, a blue arrow could equal a copy/replace/delete arrow transaction, and a green arrow could indicate a "change to" transaction, etc. The user may pick one transaction from a list of possible arrow transaction categories and assign a color to that transaction, and this relationship becomes a default for the system.
b. Power User: Assign variants of one color for various arrow transactions that are included in each arrow transaction category. For example, as shown in Figure 1, if the user designates the color blue for copy/replace/delete, the user may choose dark, default, medium and light blue hues for different types of copy/replace/delete functions.
c. Higher Power User: Assign variants of one color for various arrow transactions that are included in each arrow transaction category, plus variants of line styles for each arrow transaction category. For example, as shown in Figure 2, line styles such as thin, dashed, dotted, slotted, and solid thick line styles may be employed in addition to the various color hues of Figure 1. This approach has a lot of flexibility, depending on how many arrow transactions a user may wish to designate with a single color. For instance, the arrow option of Figures 1 and 2 may be combined to provide 16 different arrow appearances: four styles of arrows for four different hues of the color blue, and each may be assigned a unique transaction.

[0073] To relate each color hue and line style combination to a particular transaction, the user may enter the Info Window for arrow logics or for the specific arrow transaction that is to be assigned; i.e., place inside, send signal, as shown in

Figure 3. Selecting a new function for the selected color (and/or line style) for that transaction establishes the relationship, which can be immediately stored. From that point on, the selected color/line style for that arrow transaction becomes the default, unless altered by use of the Info Window once again.

[0074] For instance, if the copy/replace/delete logic color is dark blue and the transaction is: "*'Copy the definition'* of the object at the tail of the arrow to the object at the front of the arrow," one can change this transaction by selecting a new transaction from a list of possible transactions in the copy/replace/ delete Info Window. The assignment of a particular color and line style of an arrow to a particular arrow transaction can be accomplished by drawing the desired arrow (with the selected color and line style) next to the arrow logic sentence that this arrow is desired to initiate. This drawing can take place directly on the Arrow Logic Info Window page as shown in Figure 3.

[0075] NOTE: It is possible for more than one arrow color and/or line style to be assigned to a specific arrow logic. For instance, for the more common arrow transactions, i.e., "control the object and/or device that the arrow is drawn to by the object or device that the arrow is drawn from," such an arrow logic could utilize a blue arrow with a solid line and a green arrow with a solid line, etc. Similarly, it is possible to utilize a single type of arrow, i.e., a green dashed arrow, to simultaneously initiate more than one arrow transaction. To set up such an arrow logic, an arrow could be drawn on the Arrow Logic Info Window across from a specific arrow transaction. Then the same colored arrow with the same style could be drawn across from another arrow logic in the same Info Window.

[0076] NOTE: This Info Window can be found inside the Global Arrow Logic Info Window or can be entered directly. Furthermore, other methods to alter an arrow logic or assign an arrow logic include using vocal commands or typing or writing or printing text near the arrow for which its logic is to be changed or initially determined (in the case that such arrow has no logic previously assigned to it.) Another very advanced method of defining an arrow logic for an arrow would be to draw another arrow from an object that represents a particular arrow logic to an existing arrow such that the logic of the object that the arrow's tail points to is assigned to the arrow that the newly drawn arrow points to. If one selects a new transaction, i.e., " '*Copy all non-aesthetic properties*' of the object that the arrow is drawn from to the object that the arrow is drawn to," the dark blue arrow will have a new copy/replace/delete function. This function can remain until which time it is changed again.

[0077] A further line style variant that may be employed to provide further differentiation among various arrows on the graphic display is a gradient fill, as shown in Figure 4. This feature may be employed with monocolor arrows, or may gradiate from one color to another. There are several forms of gradient fills that may be used (monocolor, bicolor, light-to-dark, dark-to-light, etc.) whereby the combinations of line hues, line styles, and gradient fills are very numerous and easily distinguished on a graphic display.

[0078] Line color may be selected from an on-screen menu, as suggested in Figure 5, in which the hatching indicates different colors for the labeled buttons, and Figure 6 (not hatched to represent colors), which displays a convenient, abbreviated form of the Info Window to enable the user to select category line styles as well as shades of each color category.

[0079] **1. COPY/Replace**

[0080] This function copies all or part of any object or objects at the tail of an arrow to one or more objects at the head of the arrow. If the object that the arrow is drawn to does not have the property that a specific arrow transaction would copy or assign to it, then the arrow performs its "copy" automatically. If, however, the object the arrow is drawn to already has such property or properties, a pop up window appears asking if you wish to replace such property or properties or such object.

[0081] For example, as shown in Figure 7, one may copy the rectangle object (including all properties, i.e., Info Window, automation, aesthetic properties, definition, assignment, action and function) by drawing an arrow from the rectangle to an empty space on the screen display (i.e., a space that is not within a default distance to another screen object). Many different schemes are possible to implement the copy function. One such scheme is that the object is copied so that the front of the arrow points to either the extreme upper left corner or the upper extremity of the object, whichever is appropriate for the object, as shown by the examples of Figures 8 and 9. Copying may involve some or all the attributes of the object at the tail of the arrow; for example:

[0082] **Aesthetic Properties**

[0083] Copy the color of the object at the tail of an arrow to one or more objects at the head of the arrow.

[0084] Copy the shape of the object at the tail of an arrow to one or more objects at the head of the arrow.

[0085] Copy the line thickness of the object at the tail of an arrow to one or more objects at the head of the arrow.

[0086] Copy the size of the object at the tail of an arrow to one or more objects at the head of the arrow.

[0087] Copy all aesthetic properties (except location) of the object at the tail of an arrow to one or more objects at the head of the arrow, etc

[0088] ***Definition***

[0089] Copy the definition of the object at the tail of an arrow to one or more objects at the head of the arrow.

[0090] **Action**

[0091] Copy the action of the object at the tail of an arrow to one or more objects at the head of the arrow.

[0092] **Assignment**

[0093] Copy the assignment of the object at the tail of an arrow to one or more objects at the head of the arrow.

[0094] ***Function***

[0095] Copy the function of the object at the tail of an arrow to one or more objects at the head of the arrow.

[0096] ***Automation***

[0097] Copy the automation of the object at the tail of an arrow to one or more objects at the head of the arrow.

[0098] **Info Window**

[0099] Copy the Info Window of the object, or the contents of the Info Window, at the tail of an arrow to one or more objects at the front of the arrow.

[00100] To engage "replace", the user may click in a box for the "replace" option in the appropriate Info Window (or its equivalent) or type "replace" along the arrow stem when typing, writing or speaking a new function for a certain color and style of arrow (see also Figure 43).

[00101] Copy the object (including all properties, i.e., Info Window, automation, aesthetic properties, definition, assignment, action and function) that the arrow is drawn from and replace the object that the arrow is drawn to with such object in the location of the object that the arrow is drawn to.

[00102] Copy all non-aesthetic properties of the object at the tail of an arrow and replace those properties in one or more objects at the front of the arrow.

[00103] Copy all properties (except location) of the object at the tail of an arrow and replace those properties in one or more objects at the front of the arrow.

[00104] **Using arrow logic abbreviations.** One feature of arrow logics is that the arrow logic sentences, which can be found in arrow logic Info Windows, menus and the like, can be listed where the first number of words of the sentence are distinguished from the rest of the sentence. One way to do this is to have these first number of words be of a different color, i.e., red, and have the rest of the arrow logic sentence be in another color, i.e., black. (Note that in Figure 3, the highlighted words of the arrow logic sentences are shown in bold to indicate a color differential) The user can declare or change the logic for any given arrow (or its equivalent, i.e., a line) by typing, writing, printing or speaking just the abbreviation for the arrow logic. This shortcut eliminates the need to enter a long sentence which describes a particular arrow logic. The abbreviation performs the same task. A sample of the use of arrow logic abbreviations entered adjacent to arrow stems to assert the arrow transactions is shown in Figure 10.

[00105] **2. Place Inside**

[00106] With regard to Figure 11, the "place inside" arrow transaction enables an arrow to place a group of objects inside a folder, icon or switch or other type of hand drawn graphic. An example of this is selecting a group of sound files by drawing an ellipse around them and then drawing a line with an arrow on the end extending from the ellipse and pointing to a folder. This type of drawn arrow will place all of these encircled sound files from the list into the folder. When the arrow is drawn to the object, in this case a folder, the operation may be carried out immediately. An alternative default, which provides the user an opportunity to abort his action, association, direction, etc. caused by the drawing of his arrow, is that the object that the arrow is drawn to (the folder in this case) begins flickering, or, as may be preferred by many users, the arrow itself starts to flicker or change color, etc. The arrow then continues to flicker until it is touched. Once touched, the flickering stops, the arrow and the ellipse disappear and the transaction is completed. NOTE: In order to engage an arrow logic, it is not necessary that the arrow must disappear. However, this is often desirable, because it eliminates having a large number of arrows drawn from object to object all over a display, where the continued visibility of these arrows could interfere with a user's ability to effectively operate and control other graphics, devices, objects, text, etc., on the display. Thus, hiding "engaged" or "implemented" arrows can eliminate confusion and screen clutter, but the implementation of arrow logics is <u>not</u> dependent upon whether any draw arrow remains visible or becomes hidden.

[00107] Multiple ellipses and arrows may be able to be drawn around different objects and then one by one the objects that the arrows are pointing to, or the arrows themselves, can be touched to complete the action. By having arrows operate this way, the action of touching a flickering object or arrow can be automated to store the exact moment that the operation of the arrow has been completed for one or more objects.

[00108] Another example of a "place inside" transaction, shown in Figure 12, involves drawing an ellipse to select a group of objects (two triangles and a rhombus)

and then placing them inside another object, a star object. By double clicking on the star, the objects that have been placed inside it can be made to fly back out of the star and resume the locations they had before they were placed inside the star object. Thus, placing objects inside another object, i.e., the star of Fig 12, carries the advantage of enabling a user to draw a single object to immediately gain access to other objects. Access to these other objects can take many forms. Below are two examples:

[00109] 1) Utilizing a single object to apply the processes, features, actions, etc. of the devices contained within this single object to other objects, text, devices, etc. These other objects could represent a group of devices that can be applied to process a sound or device or object by drawing an arrow from, for example the star of Fig 12, to another object, like a sound file. But these devices may be utilized while they remain hidden inside the star, as shown in Fig 12. By drawing an arrow from the star to a sound file or vice versa (depending upon the desired signal flow), all of the processing contained within the star may be immediately applied to the sound file, although the individual processors (represented by objects inside the star) never need to be viewed. These processors can be used to process the sound file without being directly accessed. Only a connection to the star (the object containing these processors) needs to be established.

[00110] 2) Accessing the processes, features, actions, etc. of the devices contained within a single object by having them fly out of the single object. Like example one above, the objects in the star can represent processors or any type of device, action, function, etc. By doubling clicking on the star, the individual objects stored in the star may "fly out"; i.e., reappear on the display. By further clicking on each individual object, the controls for the processor that each object represents can fly out of each object and appear on screen. These controls can then be used to modify a processor's parameters. Once modified, these controls can be made to fly back into the graphic, i.e., equilateral triangle, diamond, etc. and then in turn these objects can be made to fly back into the star, as depicted in Fig 12. The underlying concept is to be able, by hand drawing alone, to gain access to virtually any level of control, processing, action, definition, etc. without having first to search for such things or having to call them up from a menu of some kind. This type of hand drawing provides direct access to an unlimited array of functions, processes, features, actions, etc. Such access can be initiated by simply drawing an object (that represents various functions, processes, features, actions, etc.) anywhere and at any time on a display.

[00111] In a further example, shown in Figure 13, a line is drawn continuously to circumscribe the first, third, and fifth fader controllers, and the arrowhead at the end of the line is proximate to a triangle object. This arrow operation selects the first, third, and fifth controllers and places them in the triangle object. Figure 13 illustrates an arrow line being used to select an object when such line circumscribes, or substantially circumscribes, an object(s) on the screen display.

[00112] In the example of Figure 14, a line is drawn continuously and includes vertices that are proximate to the first, third, and fourth fader controllers. The software recognizes each vertex and its proximity to a screen object, and selects the respective proximate objects. The arrowhead proximate to the triangle directs the arrow logic system to place the first, third, and fourth fader controllers in the triangle.

[00113] An alternate method of accomplishing this same task involves a group of objects that are selected and then dragged over the top of a switch or folder. The switch itself becomes highlighted and the objects are placed inside the switch and the switch takes on the label of the group of objects or a single object, as the case may be.

[00114] As shown in Figure 15, one or more files in a list of sound files on the screen may be chosen by drawing a line about each one and extending the arrow head to an object, in this case, a folder. As each object in the list (or group of the preceding examples) is encircled, or partially encircled, in a hand drawn ellipse, it may change color or highlight to show that it has been selected. In the example of Figure 15, only the selected text objects will be highlighted, and after the flickering folder object or arrow is touched, the selected objects will disappear from the list (or be grayed out on the list, etc.) and be placed into the folder. One value of this technique is to show which files in the list have been copied into the folder and which ones in the list remain uncopied.

[00115] With regard to Figure 44, another technique for selecting multiple objects with an arrow is the use of a "line connect mode". This mode involves drawing an arrow stem to intersect one or more objects which are thus automatically selected. These selected objects can, with the use of an arrow logic, be assigned to, sent to, summed to, etc. another object and/or device or group of objects and/or devices at the head of the arrow. In this example, the arrow associates knobs 1, 2, 6, and 7 to a single object, a triangle. The arrow transaction for this assignment is in accordance with the color, line style and or context that this arrow is drawn and according to the arrow logic assigned to that graphic combination.

[00116] Copy and Place Inside:

[00117] Place all objects at the tail of an arrow into the object at the head of the arrow. Furthermore, do not erase the original objects from the screen after they are placed in the object to which the arrow is pointing. The files that have been selected by the hand drawn ellipse and then copied to the folder become grayed out, rather than completely deleted from the list. This way the user can see that these files have been copied to an object and that the other files (the non-grayed out files) have not been copied to the folder, similar to the showing of Figure 15.

[00118] 3. Send the signal or contents to:

[00119] This arrow transaction is designed for the purpose of processing or controlling one or more objects, devices, texts, etc. with another object, text, device, etc. Thus an arrow may be drawn to send the signal of a console channel to an echo unit or send a sound file to an equalizer. This arrow transaction could also be used to send the contents of a folder to a processor, i.e., a color correction unit, etc.

[00120] Send Only

[00121] This arrow transaction sends the signal or contents of the object(s) at the tail of the arrow to the object(s) at the head of the arrow. As shown in Figure 16, one example includes a star designated as an echo chamber. By drawing the arrow from the snare sound file 'snare 1B' to the star, the snare sound signal is commanded to be sent to that echo chamber. In another example, shown in Figure 17, a triangle equals a group of console channels. The signals from these console channels are directed by one arrow to a fader, and the signals from the fader are directed by another arrow to a generic signal processing channel, which is represented by a rectangle. The actions are by no way limited to audio signals. They can be equally effective for designating control and flow between any types of devices for anything from oil and gas pipelines to sending signals to pieces of test or medical equipment, processing video signals, and the like.

[00122] NOTE: Generally, in the illustrations herein, the head and tail of an arrow must be within a default distance from an on-screen object in order to couple the transaction embodied in the arrow to the object, unless the arrow is governed by a context, which does not require a gap default of any kind. The default distance may be selectively varied in an Info Window to suit the needs of the user.

[00123] **Send/Sum**

[00124] This arrow transaction sends the signal or contents of the object(s) at the tail of the arrow to a summing circuit at the input of the object at the head of the arrow. With regard to Figure 18, one example of "send/sum" includes a pair of fader controllers, each having an arrow drawn therefrom to a third fader controller. The software may interpret the converging arrows to designate that the signals from the pair of faders are to be summed and then controlled by the third fader. Depending on default context assignments, it may be necessary to designate an arrow color for the two arrows of Figure 18 to impart the summing transaction to the third fader, otherwise two signals entering a single component may be interpreted to be ambiguous and not permissible.

[00125] As shown in Figure 19, a first arrow may be drawn from one fader controller to a second fader controller, and a second arrow may be drawn from a third fader controller to the first arrow. This construction also commands that the signals from the first and third faders are summed before being operated on by the second fader. Thus the construction of Figure 19 is equivalent to that of Figure 18.

[00126] With regard to Figure 20, the send/sum transaction may be set forth in a specific context, thereby eliminating the need for a special arrow color or appearance to direct the summing function of two inputs to a screen object. In this example, a fader is drawn on the screen, and labeled "Volume Sum" (by spoken word(s), typed label entry, etc.). The software recognizes this phrase and establishes a context for the fader. Thereafter, arrows of no special color or style may be drawn from other screen objects, such as the two other fader controllers shown in Figure 20, to the Volume Sum fader, and the signals sent to the Volume Sum fader can be added before being processed thereat. Likewise, as shown in Figure 21, the construction of Figure 19 (arrow drawn to arrow line) may be combined with a particular context (Volume Sum) to enable the send/sum transaction to occur without requiring specific arrow colors or styles. Note: The arrows shown in Figures 20 and 21 may utilize specific arrow colors and styles if these are desired by the user. Such arrows and styles may or may not be redundant, but certainly they could improve ease of use and ongoing familiarity of user operation.

[00127] **4. Change to**

[00128] One or more objects may be selected by being circumscribed by an arrow line which extends to another object. Text may be entered by voice, typing, printing, writing, etc. that states "change object to," a phrase that is recognized by the software and directed by the arrow. The transaction is that all the selected objects are changed to become the object at the head of the arrow. Thus, as shown in Figure 22, the square, ellipse and triangle that are encircled by the arrow are commanded to be changed to the star object at the head of the arrow. Note: such encircling arrow line does not have to be an enclosed curved figure. It could be a partially open figure.

[00129] The "change to" arrow transaction may also be used to alter the signal or contents of the object at the tail of the arrow according to the instructions provided for by text and/or objects at the head of the arrow. As shown in Figure 23, the two fader controllers at the left may be encircled by an arrow that is drawn to a text command that states "change to 40 bit resolution." In this case, only the two leftmost faders would be selected and modified in this manner.

[00130] 5. Specialty Arrows

[00131] Specialty arrows convey a transaction between two or more objects on screen, and the transaction is determined by the context of the arrow, not necessarily by the color or appearance of the arrow. To avoid confusion with arrows having color or appearance associated with specific transactions, the specialty arrow category may make use of a common color or appearance: e.g., the color cement gray, to designate this type of arrow. Specialty arrow transactions (arrow logics) may include, but are not limited to,

(a) Insert a modifier, action or function in the stem of an arrow;

(b) Rotational direction for a knob;

(c) Reorder the signal flow or the order of devices in a list;

(d) Apply the control of a device to one or more objects, devices or text.

(e) Create multiple copies of, and place these copies on a screen display.

(f) Exchange or swap – requires color for specific applications other than default.

[00132] As shown in Figure 24, a specialty arrow may be used to insert a modifier in an arrow stem for an existing action, function, control, etc. This technique enables a user to insert a parameter in the stem of a first arrow by drawing a second arrow which intersects the stem of the first arrow and that modifies the manner in which the first device controls the second. In this case, the user inserts a specifier in an arrow stem to, for example, alter the ratio or type of control. The inserted '.5' text conveys the command that moving the fader a certain amount will change the EQ1 control by half that amount.

[00133] In order to enter a specialty arrow as in Figure 24 (or elsewhere herein) it may be necessary to use the "Show Control" or "Show Path" command, or its equivalent, to make visible the arrows that have been previously activated and thereafter hidden from view. This "Show" function may be called forth by a pull-down menu, pop-up menu, a verbal command, writing or printing, or by drawing a symbol for it and implementing its function. For example, the circle drawn within an ellipse, which may represent an eye, may be recognized as the Show Path or Show

Arrow command. Once this object is drawn, a graphic which shows the control link between the fader and the knob will appear. At this point, the user may draw an arrow that intersects this now visible link between the fader and the knob to create a modification to the type (or ratio) of control. There can be a system default stating that a 1:1 ratio of control is implied for any arrow transaction; thus, for example, for a given amount of change in the fader, that same amount of change is carried out in the knob, which is being controlled by that fader. But the addition of the arrow modifier extending from the .5 symbol modifies the relationship to 2:1; that is, for a given amount of change in the fader, half that much change will occur in the knob that is being controlled by that fader.

[00134] Alternatively, the modifying arrow may be entered when the first arrow is drawn (from the fader to the knob in Figure 24) and begins to flicker. The second, modifier arrow may be drawn while the first arrow is flickering, and the two arrows will then flicker together until one of them is touched, tapped, or clicked on by a cursor, causing the arrow transactions to be carried out.

[00135] In either case, the context of the second modifier arrow is recognized by the arrow logic system. The second arrow is drawn to a first arrow, but the second arrow does not extend from another screen object, as in Figures 19 or 21; rather, it extends from a symbol that is recognized by the system to impart a modifier to the transaction conveyed by the first arrow. Thus the context determines the meaning of the arrow, not the color or style of the arrow.

[00136] With regard to Figure 25, the context of an arrow may be used to determine the conveyance of an action or function. This technique enables a user to insert another device, action, function etc., in the stem of an arrow by drawing a second arrow which points to (is within a gap default), or intersects the stem of the first arrow and which extends from the inserted device, action, function, etc. In this example, the volume fader is interposed between the drum kit 1B signal source and the triangle, which may represent a signal processing function of some defined nature, so that the fader adjusts the volume of the signal that is transferred from the drum kit 1B folder to the triangle object. A default of this approach which is protective to the user may be that the inserted arrow must be the same color as the first arrow. On the other hand, a context may be used to determine the transaction, regardless of arrow color or style. The context can be the determining factor, not requiring a special color and/or line style to denote a particular arrow logic, namely: insert whatever device is drawn at the tail of the arrow, which is pointing to the existing arrow stem.

[00137] NOTE: Color can be used to avoid accidental interaction of arrows. For instance, arrow lines which are not the same color as existing lines may be draw across such existing lines without affecting them. In other words, it can be determined in software that drawing another arrow, that intersects with an existing arrow, will not affects the first arrow's operation, function, action, etc., unless the second arrow's color is the same as the first arrow's color. In this case, by choosing a different color, one can ensure that any new arrow or object drawn near or intersecting with an existing arrow or object will avoid any interaction with the existing arrow or object. Default settings in the arrow logic system can specify the conventions of color used to govern these contexts.

[00138] It is noted that other non-arrow methods may be used to impart an action or function or modifier to screen objects. As shown in Figure 26, once a fader or other controller is drawn on a screen display, a control word such as "Volume" may be spoken, typed or written into the system at a location proximate to the fader. The system then recognizes the word and imparts the function 'volume' to the adjacent fader. Another implementation of this idea is shown in Figure 27, where typing, writing or speaking the entry "0.0dB" proximate to the existing fader accomplishes two things: 1) It determines the resolution and range of the device (fader). For example, "0.0" establishes control of a variable to tenths of dB, and a range of 0.0-9.9. If "0.00" were entered, this would mean hundreds of dB control, etc.; 2) It determines the *type of units* of control that the device (fader) will operate with. In the case of this example, "dB" or decibels is the unit. If "ms" (milliseconds) were designated, then this device's units would be time. If "%" (percent) were entered, then this device's units would be percent, etc.

[00139] An additional embodiment of this idea can be seen in Figure 28, where the entry of the scale factors "+10 dB" and "-10 dB" proximate to and placed along the track of a fader controller causes not only the fader to be recognized as a dB controller, but also that the fader's scaling is user defined. That is, the distance between the +10dB text and the –10dB text defines the scaling for this fader device. In other words, it defines the rate of dB change for a given distance of fader movement – the movement of the fader cap along the fader track. Therefore, the distance between the ±10 dB labels corresponds to the fader cap positions that in turn yield the labeled control (up 10 dB or down 10 dB). This context-driven function entry also may also cause a label "10 dB" to be placed at the top of the fader track.

[00140] A scale factor may be applied in the same manner to a knob controller, as shown in Figure 29, with the angular span between the scale labels representing ±10 dB range of the knob controller.

[00141] With regard to Figures 30 and 31A, specialty arrows may be used to indicate the direction of rotation of a knob controller (or translation of a fader cap's movement). The context elements (curved arrow, drawn proximate to a knob controller), create a relationship in which the knob function increases with clockwise rotation (toward the head of the arrow), and the arrow of Figure 31A specifies a counterclockwise increase in knob function. However, it is possible to overcome any defined action, as shown in Figures 32 and 33, by entering the nature of the function change as the knob is rotated in the arrow direction. Figure 32 specifies negative change in the clockwise direction, and Figure 33 specifies negative change in the counterclockwise direction, both the opposite of Figures 30 and 31.

[00142] This example raises another instance in which the system is designed to be context-sensitive. With reference to Figure 31B, the curved arrow drawn between two knob controllers may appear to be ambiguous, since it is sufficiently proximate to both screen objects to be operatively associated with either one. However, the curvature of the arrow may recognized by the arrow logic system (through processes described in the parent application referenced above), and this curvature is generally congruent with the knob on the right, and opposed to the curvature of the knob on the left. Alternatively, the system may recognize that the curved arrow partially circumscribes the right knob, and not the left knob. In either case, the context determines that the arrow transaction is applied to the knob on the right.

[00143] Specialty arrows may further be used to apply the control function of a device to one or more objects, devices, text, etc. When it is utilized in a context situation, the color or line style of the arrow is not necessarily important. Any color or line style may work or the one color (gray) specified above for context arrows may be used. The important factor for determining the control of an unlabeled device (fader, knob, joystick, switch, etc.) is the context of the hand drawn arrow drawn from that device to another device. As shown in Figure 34A, drawing an arrow from a functional fader (a fader with a labeled function, i.e., Volume) to another object, in this case a folder that contains a number of sound files, will automatically apply the function of that fader, knob, joystick, etc. to the object to which it is drawn. In this case, the context is "controlling the volume of". There can be no other interpretation for this device (fader). It is a volume fader, so when an arrow is drawn from it to a folder containing a group of sound files, the fader controls the volume of each sound file in the folder. As in all the previous examples, the arrow transaction is invoked if the tail of the arrow is within a default distance to any portion of the fader controller screen object, and the head of the arrow is within a default distance of the folder containing the sound files.

[00144] In a further example, shown in Figure 34B, a pair of fader controllers are arrow-connected to respective left and right tracks of sound file "S: L-PianoG4-R". The context of the two fader controllers linked by respective arrows to the left and right sides of the text object is interpreted by the software to indicate that each fader controls the respectively linked track of the stereo sound file.

[00145] A further use for specialty arrows is to reorder a signal path or rearrange the order of processing of any variable in general. As shown in Figure 35, reordering can involve drawing an ellipse or an intersecting line or a multiple vertex line to select a group of devices and then drawing an arrow from this selected group of devices, which can be functional devices, to a new point in a signal path. When the arrow is drawn, it may start to flicker. Touching the flickering arrow completes the change in the signal path.

[00146] Note: if a user is familiar with this system and is confident about using arrows, the use of flickering arrows or objects may be turned off. In this case, when an arrow is drawn, the action, function, etc. of that arrow would be immediately implemented and no flickering would occur. Needless to say, any such arrow action could be aborted or reversed by using an undo command or its equivalent. The arrow of Figure 35 moves the Rich Chamber echo to the input of the EQ 3B (a general signal processing device). This change in signal path causes the signal to flow first into the echo Rich Chamber and then into the EQ 3B.

[00147] In another example, shown in Figure 36, a curved line is drawn about the volume control, with an arrow extending therefrom to the input of EQ 3B. This arrow transaction commands that the volume control function is placed at the input of the EQ, whereby the input to the EQ 3B is first attenuated or increased by the volume control. Likewise, drawing an arrow from the volume label to intersect the label "EQ 3B", as shown in Figure 37, applies the volume control function of the knob controller to the input signal of the EQ. In a further example, shown in Figure 38, an arrow is drawn from one fader controller, to and about the Rich Plate echo control, and then to the input of EQ 3B. The direction and connections of this arrow commands that the output of the leftmost fader (at the tail of the arrow) is fed first to the Rich Plate echo control, and then to the input of EQ 3B at the head of the arrow.

[00148] In the examples of Figures 35-38, the contexts of the drawn arrows determine the transactions imparted by the arrows; that is, an arrow drawn from one or more controllers to another one (or more) controllers will direct a signal to take that path. This context may supercede any color or style designations of the drawn arrows, or, alternatively, may require a default color as described in the foregoing specification.

[00149] Another use of specialty arrows is to create multiple copies of screen objects, and place these copies on a screen display according to a default or user defined setup. This feature enables a user to create one or more copies of a complex setup and have them applied according to a default template or according to a user defined template that could be stored in the Info Window for this particular type of action. For example, as shown in Figure 39, a combination of functional screen objects, such as a fader controller, and a triangle, circle, and star, any of which may represent functional devices for signal processing, are bracketed and labeled "Channel 1". For instance, the triangle could equal a six band equalizer; the circle, a compressor/gate; and the star, an echo unit. An arrow is then drawn from the Channel 1 label to an empty space on the screen. As the arrow flashes, the stem of the arrow is modified by the input (spoken, written or typed) "Create 48 channels." The system interprets this instruction and arrow as a command to produce 48 channels, all of which have the construction and appearance of Channel 1. The action indicated is: "Copy the object that the arrow is drawn from, as many times as indicated by the text typed near the arrow stem pointing to blank space. Furthermore, copy this object according to the default template for console channels." The default may be, for example, place 8 console channels at one time on the screen and have these channels fill the entire available space of the screen, etc. The specialty arrow is once again controlled by context, not by color or style: the tail of the arrow is proximate to an object or group of objects, the head of the arrow is not proximate to any screen object, and the arrow is labeled to make a specified number of copies. Note that the label of the arrow may simply state "48" or any other suitable abbreviation, and if the system default is set to recognize this label as a copy command, the arrow transaction will be recognized and implemented.

[00150] A further specialty arrow is one used to exchange or swap one or more aspects of two different screen objects. The arrow is a double headed arrow that is drawn between the two objects to be involved in the exchange transaction. This double headed arrow head creates a context that can only be "swap" or "exchange". The other part of the context is the two objects that this double headed arrow is drawn between.

[00151] To facilitate recognition of a wide range of screen objects, the system may provide a default that the double headed arrow must be drawn as a single stroke.

As shown in Figure 40A, the start of the arrow (at the left) is a half arrowhead and the end of the arrow is a full arrowhead. This is a very recognizable object that is unique among arrow logics and contextually determinative. Once recognized, the drawn arrow is replaced by a display arrow (Figure 40B) that can flicker until touched to confirm the transaction. The list of aspects that may be swapped has as least as many entries as the list given previously for possible copy functions:

[00152] Aesthetic Properties

[00153] Swap the color of the object, the shape of the object, the line thickness of the object, the size of the object, or all aesthetic properties (except location) between the objects at the head and tail of the arrow.

[00154] ***Definition***

[00155] Swap the definitions of the objects at the head and tail of the arrow.

[00156] ***Action***

[00157] Swap of action of the objects at the head and tail of the arrow.

[00158] ***Assignment***

[00159] Swap of assignment of the objects at the head and tail of the arrow.

[00160] ***Function***

[00161] Swap the function of the objects at the head and tail of the arrow.

[00162] ***Automation***

[00163] Swap the automation of the objects at the head and tail of the arrow.

[00164] **Info Window**

[00165] Swap the Info Window of the objects at the head and tail of the arrow.

[00166] The technique for arrow entry of Figure 39, shown in Figures 41-43, involves the initial drawing of an arrow, as shown in Figure 41, followed by the presentation of a flickering arrow on the display (Figure 42). Thereafter, the user may place a text cursor within a default distance to the flickering arrow (Figure 43), and speak, write or type a simple phrase or sentence that includes key words recognized by the software (as described with reference to Figure 3). These words may be highlighted after entry when they are recognized by the system. As previously indicated in Figure 39, the recognized command of the phrase or sentence is applied to the adjacent arrow, modifying the transaction it conveys. In addition, as an extension of this technique, the user may first enter the phrase or sentence that expresses the desired transaction on the screen, and then draw an arrow within a default distance to the phrase or sentence, in order for the arrow and text command to become associated. Likewise, typing or speaking a new command phrase or sentence within a default distance of an existing arrow on-screen may be used to modify the existing arrow and alter the transaction conveyed by the arrow. Note: a spoken phrase would normally be applied to the currently flickering arrow.

[00167] With regard to Figure 45, the arrow logic system programming may recognize a line as an arrow, even though the line has no arrow head. The line has a color and style which is used to define the arrow transaction. Any line that has the exact same aesthetic properties (i.e., color and line style) as an arrow may be recognized by the system to impart the transaction corresponding to that color and line style.

[00168] As shown in Figure 46, any shape drawn on a graphic display may be designated to be recognized as an arrow. In this Figure, a narrow curved rectangular shape drawn between a star object and a rectangle object is recognized to be an arrow that imparts a transaction between the star and the rectangle.

[00169] Definitions

[00170] Arrow: An arrow is an object drawn in a graphic display to convey a transaction from the tail of the arrow to the head of the arrow. An arrow may comprise a simple line drawn from tail to head, and may (or may not) have an arrowhead at the head end. The tail of an arrow is at the origin (first drawn point) of the arrow line, and the head is at the last drawn point of the arrow line. Alternatively, any shape drawn on a graphic display may be designated to be recognized as an arrow. The transaction conveyed by an arrow is denoted by the arrow's appearance, including combinations of color and line style. The transaction is conveyed from one or more objects associated with the arrow to one or more objects (or an empty spaced on the display) at the head of the arrow. Objects may be associated with an arrow by proximity to the tail or head of the arrow, or may be selected for association by being circumscribed (all or partially) by a portion of the arrow. The transaction conveyed by an arrow also may be determined by the context of the arrow, such as the type of objects connected by the arrow or their location. An arrow transaction may be set or modified by a text or verbal command entered within a default distance to the arrow, or by one or more arrows directing a modifier toward the first arrow. An arrow may be drawn with any type of input device, including a mouse on a computer display, or any type of touch screen or equivalent employing one of the following: a pen, finger, knob, fader, joystick, switch, or their equivalents. An arrow can be assigned to a transaction.

[00171] **Arrow configurations:** An arrow configuration is the shape of a drawn arrow or its equivalent and the relationship of this shape to other graphic objects, devices and the like. Such arrow configurations may include the following: a perfectly straight line, a relatively straight line, a curved line, an arrow comprising a partially enclosed curved shape, an arrow comprising a fully enclosed curved shape, i.e., an ellipse, an arrow drawn to intersect various objects and/or devices for the purpose of selecting such objects and/or devices, an arrow having a half drawn arrow head on one end, an arrow having a full drawn arrow head on one end, an arrow having a half drawn arrow head on both ends, an arrow having a fully drawn arrow head on both ends, a line having no arrow head, and the like. In addition, an arrow configuration may include a default, gap which is the minimum distance that the arrow head or tail must be from an object to associate the object with the arrow transaction. The default gap for the head and tail may differ.

[00172] **Arrow logic:** A transaction conveyed by an arrow (as defined herein).

[00173] **Show Arrow command:** Any command that enables a previously disappeared arrow to reappear. Such commands can employ the use of geometry rules to redraw the previous arrow to and from the object(s) that it assigns its arrow logic to. The use of geometry rules can be used to eliminate the need to memorize the exact geometry of the original drawing of such arrow.

[00174] A method for showing or illustrating assigned relationships, such as arrow logic relationships, between graphic objects on the screen in accordance with an embodiment of the invention is now described. As stated above, arrows drawn to assign various arrow logics, which include operational controls, between graphic objects disappear once the arrow logics are assigned. Since the arrows disappear, the assigned arrow logics between the objects are not readily apparent. Thus, in the arrow logic system, the arrow logics or the arrow logic relationships between graphic objects can be made to be shown by displaying graphic directional indicators, such as arrows or any other comparable graphics, to indicate the assigned arrow logics.

[00175] This method for illustrating arrow logic relationships between graphic objects is described herein with reference to a Blackspace computer operating environment in which the arrow logic system may be implemented. The word "Blackspace" is a trademark of the NBOR Corporation. Blackspace environment presents one universal drawing surface that is shared by all graphic objects within the environment. Blackspace environment is analogous to a giant drawing "canvas" on which all graphic objects generated in the environment exist and can be applied. Each of these graphic objects can have a user-created relationship to any or all the other objects. There are no barriers between any of the objects that are created for or exist on this canvas. However, this method for illustrating arrow logic relationships between objects is not limited to the Blackspace environment and can be implemented in any computer operating environment.

[00176] In the Blackspace environment, connections between graphic objects in the graphic user interface (GUI) are maintained by linkers, which are lists of graphic objects. The linkers provide various ways in which these objects may interact with each other. An example is a "glue linker" that allows objects in the linker to be moved and resized on the screen as a group. Another example of a linker is the arrow logic linker. The distinguishing feature of arrow logic linkers is that these linkers are directional. Some objects are designated as sources and others as targets. This relationship is easily determined by a user by the way the user draws the arrow that is used to create the linker. Arrow logic linkers are further categorized into different types of functional behavior by virtue of the color of the arrow used to create the arrow logic linkers and any modifying text that may be entered by the user.

[00177] Some arrow logic linkers are retained in memory after the initial user interaction is completed, while other arrow logic linkers are discarded as soon as their task is completed. An example of an arrow logic linker that is discarded is the assignment arrow logic linker. Once the assignment has been created and allocated to its owner object, there is no further need for the arrow logic linker to remain. An example of an arrow logic linker that is retained is the control arrow logic linker, which can be used, for example, to link two faders to each other such that one of the faders is controlled by the other fader.

[00178] In order for the user to view and interact with an arrow logic linker that is retained in memory, a temporary graphic called an "arrow logic display" can be called up to illustrate the connections stored in the arrow logic linker.

[00179] The method for illustrating arrow logic relationships is further described using the following example of faders. However, the described method can be applied to any source or target graphic object associated with any type of arrow logic that is retained in memory, including any other assigned relationships between graphic objects. In Figure 47A, three faders 10, 12 and 14 are shown. Arrow logics can be assigned between these faders 10, 12 and 14 such that the fader 10 controls the fader 12, which in turn controls the fader 14. Thus, the fader 10 controls the fader 14 via the fader 12. In this example, a specialty "control" arrow logic is used to assign one fader control over another fader. The control assignments between the faders 10, 12 and 14 can be achieved by drawing a first "control" arrow 16, e.g., a red arrow, from the fader 10 to the fader 12 and drawing a second "control" arrow 18 from the fader 12 to the fader 14, as illustrated in Figure 47B. The first and second control arrows 16 and 18 are then activated to assign the desired arrow logics, which in this case is the control arrow logic for each of the arrows 16 and 18. In an exemplary embodiment, the control arrows 16 and 18 are activated by touching the head of each arrow by a mouse click, a stylus on a touch screen display or by any other comparable means. After the arrow logics have been assigned, the control arrows 16 and 18 disappear, as shown in Fig. 47C. Thus, after the arrow logics having been assigned, there are no visual indicators showing the relationships between the faders 10, 12 and

14 that have been assigned by the arrow logics, until a show arrow command is entered for one or more faders.

[00180] A show arrow command can be entered by selecting an entry in a menu, e.g., a "Show Arrows" entry 22 in an Info Canvas object 20 for one of the faders 10, 12 and 14, as illustrated in Fig. 48. The term "Info Canvas" is a trademark of NBOR Corporation. An Info Canvas object (or "Info Window" as referred to above in this disclosure) for a fader provides entries to change the properties of the fader or control functions associated with the fader. Thus, the Info Canvas object for a fader serves as a menu for using that fader. For more information about Info Canvas objects, see simultaneously filed U.S. patent application serial no. xx/xxx,xxx, entitled "Intuitive Graphic User Interface with Universal Tools", which is incorporated herein by reference.

[00181] In the above example, the arrow logics associated with the fader 10 can be shown by activating the "Show Arrows" entry in the Info Canvas object for the fader 10. When the "Show Arrows" entry is activated, a red display arrow 24 is drawn (i.e., made to appear on the screen) from the fader 10 to the fader 12, as shown in Fig. 49A, to indicate that a control arrow logic is assigned from the fader 10 to the fader 12. Since there is only one arrow logic associated with the fader 10, only a single arrow is shown.

[00182] Similarly, the arrow logics associated with the fader 12 can be shown by activating the "Show Arrow" entry in the Info Canvas object for the fader 12. For the fader 12, when the "Show Arrow" entry is activated, a first red display arrow 24 is drawn from the fader 10 to the fader 12 and a second red display arrow 26 is drawn from the fader 12 to the fader 14, as shown in Fig. 49B, to indicate that a first control arrow logic is assigned from the fader 10 to the fader 12 and a second control arrow logic is assigned from the fader 12 to the fader 14. However, if one of the red display arrows was already on the screen, e.g., the red display arrow 24, then only the other display arrow is drawn. Thus, a display arrow representing the arrow logic between two objects can be made to appear by activating the show arrow command for either of the two objects.

[00183] Similarly, the arrow logics associated with the fader 14 can be shown by activating the "Show Arrows" entry in the Info Canvas object for the fader 14. For the fader 14, when the "Show Arrow" entry is activated, a single red display arrow 26 is drawn from the fader 12 to the fader 14, as shown in Fig. 49C, since the fader 14 is associated with only one arrow logic. If the red display arrow 26 was already on the screen, then the activation of the "Show Arrow" entry does not produce any change on the screen.

[00184] Turning now to Figure 50, a flowchart of a process for showing one or more display arrows to illustrate arrow logics for a given graphic object is shown. At step 28, message is received that the "show arrow" entry in the Info Canvas object of the object has been activated. In the Blackspace environment, right mouse button clicking on any graphic object causes an Info Canvas object for that object to be displayed. When an entry in an Info Canvas object is clicked on, an appropriate functional method in the graphic object is executed. Conceptually, this can be viewed as though a message from the Info Canvas object is received by the graphic object.

[00185] Next, at step 30, a determination is made whether the object has displayable links. This step is a routine that checks the list of linkers maintained in a graphic object and decides if any of them are appropriate for being illustrated to the user by the use of a display arrow. There are two lists in each graphic object. One contains all the linkers for which the object is a source. This includes all linkers that are not directional, as well as arrow logic linkers for which the object is not a target. The other list of linkers contains all the linkers for which the object is a target (only arrow logic linkers have targets). The routine looks through each of these lists in turn trying to find linkers that are functional. In this context, a functional linker is a linker that maintains controlling or other non-graphical connections and the user has no other way to view the members of the linker. This is determined by checking to see if the linker is a particular type, for example, a "send to" linker. An example of a linker that is not regarded as functional in this context would be a "graphic linker", which is the type used to maintain the objects belonging to another object. If either list contains such a functional linker, then the routine returns a value indicating that this object does contain at least one displayable linker.

[00186] This determination step 30 is now described in detail with reference to the flowchart of Figure 51A. At step 38, a linker is selected from a list of linkers for which the object is a source. Next, at step 40, a determination is made whether the selected linker is a functional linker. If yes, then it is determined that the object does have displayable links, at step 52, and the process proceeds to step 32 in the flowchart of Figure 50.

[00187] If the selected linker is determined not to be a functional linker, at step 40, then the routine proceeds to step 42, where another determination is made whether the selected linker is the last linker in the list of linkers for which the selected object is a source. If the selected linker is not the last linker, then the routine proceeds back to step 38, where then next linker in the list of linkers is selected and steps 40 and 42 are repeated. However, if the selected linker is the last linker, then the routine proceeds to step 44, where a linker is selected from a list of linkers for which the object is a target.

[00188] Next, at step 46, a determination is made whether the selected linker is a functional linker. If yes, then the routine proceeds to step 52. If no, then the routine proceeds to step 48, where another determination is made whether the selected linker is the last linker in the list of linker for which the object is a target. If the selected linker is not the last linker, then the routine proceeds back to step 44, where the next linker in the list of linkers is selected and steps 46 and 48 are repeated. However, if the selected linker is the last linker, then it is determined that the object does not have displayable links, at step 50, and the entire process comes to an end.

[00189] Referring back to Figure 50, at step 32, a linker is selected from the list of linkers to which the object belongs. In the first instance, the selected linker is the first linker in this list of linkers. Next, at step 34, a display arrow representing this linker is shown. Each linker can display a simplified graphical arrow representing the connections managed by the linker, which is now described with reference to the flowchart of Figure 51B.

[00190] Figure 51B describes a routine in the arrow logic linker, which displays a graphical representation of itself. At step 54, the list of objects in this linker is examined. Next, at step 56, a list of points representing the center of each of these objects as viewed on the global drawing surface is made.

[00191] Next, at step 58, the color of the arrow that was used to create this linker is retrieved. This step is to determine the color that the user employed to draw the arrow that created this linker. This information was saved in the data structure of the linker. Next, at step 60, a line is drawn joining each of the points in turn using the determined color, creating linear segments defined by the points. Next, at step 62, an arrowhead shape is drawn pointing to the center of the target object at an angle calculated from the last point in the sources list. In other words, an arrowhead is drawn at the same angle as the last segment so that the tip of the arrowhead is on the center of the target object in the linker.

[00192] Next, at step 64, the collection of drawn items (i.e., the line and the arrowhead) is converted into a new graphic object, referred to herein as an "arrow logic display object". Next, at step 66, the "move lock" and "copy lock" for the arrow logic display object are both set to ON so that the user cannot move or copy this object. Next, at step 68, an Info Canvas object for the arrow logic display object having only "hide" and "delete logic" entries is created.

[00193] Note: if the user moves any graphic object in the linker, then this same routine, as described in Figure 51b, is called again to redraw the arrow logic display object. After step 68, the process then proceeds to step 36 in the flowchart of Figure 50.

[00194] Referring back to Figure 50, at step 36, a determination is made whether the current linker is the last linker in the list of linkers. If no, then the process proceeds back to step 32, where the next linker in the list of linkers is selected and steps 32 and 34 are repeated. If the current linker is the last linker, then the process comes to an end.

[00195] Turning now to Figure 52, a flowchart of a process called in the arrow logic display object when the delete command is activated for the display object. At step 70, the arrow logic display object receives a delete command from its Info Canvas object. Next, at step 72, the arrow logic display object finds the linker that this display object is representing. The arrow logic display object made a note of this at the time the display object was created. Next, at step 74, the linker is deleted from the GUI system. The deletion of the linker from the GUI system causes the graphic objects to lose any functional connections with each other that are provided by the arrow logic linker. This does not cause the graphic objects to be deleted, but the affected graphic objects lose this linker from their own lists and thus cannot use the linker to perform any control or other operation that requires the capabilities of the linker.

[00196] Next, at step 76, a message is sent to all the contexts informing them that the linker has been deleted and no longer exists in the GUI. Next, at step 78, contexts will remove any functional connections that the creation of the linker initiated. In the case of a linker, there may be contexts that have set up some other (non-GUI) connection(s) at the time the linker was created. These associations are disconnected at step 78.

[00197] A method for illustrating assigned relationships, e.g., arrow logic relationships, between graphic objects in accordance with an embodiment of the invention is described with reference to a flow diagram of Figure 53. At step 80, first and second graphic objects are displayed. The first graphic object has assigned relationship with the second graphic object. Next, at step 82, a graphic directional indicator, e.g., a display arrow, between the first and second graphic objects is displayed in response to a user input requesting assigned relationship between the first and second graphic objects to be shown.

[00198] In an embodiment of the invention, the method for illustrating assigned relationships, e.g., arrow logic relationships, between graphic objects is performed by a computer program running in a computer. In this respect, another embodiment of the invention is a storage medium, readable by a computer, tangibly embodying a program of instructions executable by the computer to perform the method steps for illustrating assigned relationships between graphic objects.

[00199] Although specific embodiments of the invention have been described and illustrated, the invention is not to be limited to the specific forms or arrangements of parts so described and illustrated. The scope of the invention is to be defined by the claims appended hereto and their equivalents.